תרגיל בית 3

234124 - מבוא לתכנות מערכות

שם: פרח קב

ת״ז: 213309388

שם: ג׳מאל אבו מוך

ת״ז: 213754088

חלק יבש 3

3.1 שאלות על SortedList

1. במבנה SortedList הסבירו במשפט T הגנרי, מה הדרישות ההכרחיות שעל הטיפוס לקיים? לכל דרישה

In order to store the data independently form the user's memory course we need to copy the data, therefore a copy constructor on the T data type is required.

In conclusion it cannot be a deleted constructor.

To adhere to the ordering requirement, sorting mechanism must be established.

In our case the less than operator is used for comparison between the data, therefore the T data type must implement a less than operator.

2. כלומר, עבור איטרטור SortedList-עבור ה non-constמניח כי היינו רוצים לממש איטרטור. איזו בעיה עלולה להיווצר במימוש .T&האופרטור* היה מחזיר, זה?

If the **SortedList** was const, then the non-const iterator, provided that holder offers an access to data without const modifier, can return a **T&** reference to data, even when the **SortedList** in itself is const, therefore the possibility of modifying the data, which is an ill behavior.

To counter this issue, both **Iterator** class and **ConstIterator** can be defined with the proper signature.

3. סטודנטית בקורס מבוא לתכנות מערכות סיימה לפתור את תרגיל בית 3 והחליטה להשתמש ברשימה הממויינת מהתרגיל לפרוייקט צד שהיא מפתחת בשעות הפנאי. במימוש פרוייקט הצד הסטודנטית נדרשה לסנן רשימה של מספרים שלמים, כך שישארו בתור רק מספרים המתחלקים במספר כלשהו שאינו ידוע בזמן קומפילציה אלא רק בזמן ריצה. הסבירו כיצד ניתן למשש את הפונקציונליות הדרושה בעזרת הפונקציה filter.

In the typical way defining the signature by a pointer to a function, all functions passed must be explicitly defined in compile time or created dynamically as a pointer to a function, which is a complicated and not memory safe option, ergo, templated parameters can be used to pass a function or a lambda, stateful or not, which is a very convenient way to allow inclusion of data types, which allows, defining a class with an overloaded operator (), that can have a state per instance which is the solution to runtime-only known data.

A signature with **const Func&** parameter type can also be implemented to allow handlers that do not perform side-effects, but it's dependent on the application requirements.

Example:

```
template<class Func>
SortedList filter(Func& predicate) const {
            ……
}
```

```
class TaskFilterByTypeHandler {
private:
    TaskType type;
public:
    TaskFilterByTypeHandler(TaskType type);

    bool operator()(const Task &);
};
```

```
TaskFilterByTypeHandler::TaskFilterByTypeHandler(TaskType type) :
type(type) {

}

bool TaskFilterByTypeHandler::operator()(const Task &task) {
    return task.getType() == type;
}
```

Usage with SortedList::filter:

```
TaskFilterByTypeHandler handler(type);

list.filter(handler);
```